Memory Overcommit… without the commitment

Dan Magenheimer, Oracle Corp.

Extended Abstract for Xen Summit, June 2008

Introduction

Valuable resources in physical servers are often underutilized and virtualization can be employed to consolidate multiple physical servers so that resources are more fully utilized. If a VMM creates the illusion that there are more instances (or bandwidth) of a resource than actually physically available, that resource is considered to be *overcommitted* (or oversubscribed). For example, if three uniprocessor servers are virtualized (as *guests*) and consolidated onto one dual-processor server, the CPU resource is considered overcommitted. If all three guests each need to run workloads that always require an entire CPU, these guests would be considered poor candidates for virtualization. But, virtualization works most of the time because most resources on many physical servers are badly underutilized.

One key resource however is a notable exception: physical memory. Unlike CPUs and I/O cards, an operating system on a physical system generally utilizes its physical memory fully. Pages of memory that are not used for kernel/application code or data are used to cache disk data. Since disks are extremely slow and it is common for the same disk data to be used again and again, a well-managed cache of disk pages can dramatically improve performance of many workloads. Consequently, when physical servers are consolidated, it is considered unwise to overcommit physical memory. Indeed, Xen does not support memory overcommitment at all. For example, if one wishes to consolidate multiple 1GB guests on a 4GB Xen server, only three guests can be accomodated.

Memory Overcommitment Elsewhere

Some Vmware products optionally support memory overcommit. Using black-box mechanisms described in [10], up to five times as many guests can be run when memory overcommit is enabled. In a recent blog posting [4], Vmware exploits this capability to argue that their products are less expensive than Xen or Microsoft Virtual Server as measured by "cost-per-virtual-machine". Many responses to this posting, argued:

- Real world workloads don't benefit
- Memory is cheap: just add more physical memory
- Vmware doesn't recommend turning it on anyway
- Migration doesn't work with memory overcommitment

But the truth is that many workloads *do* benefit from memory overcommit, additional memory isn't a panacaea, Microsoft is working on it [8], and so is kvm [5]. So why not Xen? Making it maximally efficient is very hard and (even with Vmware) can lead to stability issues, functionality limitations, and some loss of performance. But can Xen get most of the benefit, without incurring these costs?

Memory Overcommitment Mechanisms

Waldspurger in [10] describes three VMware techniques:

- *Ballooning* utilizes a special driver that "inflates" by requesting memory from the kernel and returning it to the VMM. Xen provides a functioning balloon driver and some limited tools for interacting with it but no management. We will return to ballooning shortly.
- *Content-based page sharing* transparently discovers identical pages and combines them, using copy-on-write to separate them again when necessary. This has been explored on Xen in [6] and [7]. It is complex to implement, has limited benefit in non-idle domains, and has a performance impact.
- In *demand paging*, the VMM maintains a swap area and can page out memory without guest involvement. Because of the so-called "semantic gap" [2], a VMM-based page replacement algorithm can't be efficient and can lead to double paging. Since Xen has no I/O capability in the hypervisor, domain0 would need to manage a system-wide swap area, which leads to even more complexity and inefficiency.

[9] suggests that memory hotplug could be used as an underlying mechanism and [3] suggests that entire guests might be swapped out -- just as entire processes were swapped in early operating systems; guests are swapped in when triggered by something like wake-on-LAN.

Is Xen Ballooning Enough?

Just as with CPUs and I/O, the physical memory in most physical servers may also often be underutilized. When a server has idle CPUs, much of physical memory is holding stale application data or disk pages that may never be used again. Today's operating systems do a fine job of optimizing memory usage by retaining pages that *might* be used again but no operating system can accurately predict the future, so some -- and in many cases most -- of its guesses are wrong. As a result, much of the physical memory is *idle memory* that is just taking up space and

won't be used again. In other words, physical memory may be *fully* utilized but it is often not *efficiently* utilized.

A balloon driver, when properly managed, complements nicely with the Linux kernel's page replacement mechanisms. When the balloon is inflated by one page, the kernel surrenders the page that it believes is least likely to be used again. If the balloon can be inflated until just before the guest "hurts" for memory, the guest's idle memory will be minimized, and Xen can use that otherwise idle memory for another guest. When a guest needs more memory, it must be able to quickly deflate the balloon; but if additional memory is unavailable, a properly configured Linux guest swaps pages to disk, just as if it reached a fixed limit of physical memory. Then when memory later becomes available, Linux can make use of that memory to swap pages in from disk if it needs to.

In [11], gray-box data is collected to manage migration strategies. By using a similar approach to ballooning, data provided by Linux itself can be used to control balloon inflation and deflation. Specifically, the *Committed_AS* statistic in /proc/meminfo provides a reasonable estimate of memory "need" that is conservative, reasonably accurate, and dynamic. By using this value to inflate and deflate the balloon automatically, a process we call *selfballooning*, idle memory appears to be roughly minimized.

What about out-of-memory (OOM) conditions? Assuming Linux has basically enough memory and/or is given enough time to adapt to low-memory conditions, OOMs don't happen. A minimum memory value (varying depending on total memory available) was recently added [1] to the Xen balloon driver to ensure the balloon doesn't inflate too much. And including hysteresis in the ballooning algorithm, balloon inflation can be rate-limited so that memory is never reduced too quickly for Linux to adapt.

Implementation

At first, we tried making changes in the Xen balloon driver itself. Because some key memory statistic variables are not exported from the kernel (even though they are available in /proc), we decided to move all balloon management into user-mode. We implemented, *xenballoond*, a guest-resident service (daemon) written entirely in bash script that supports both selfballooning and "directed ballooning". For selfballooning, the guest need only have the balloon driver installed either in-kernel (paravirtualized) or as a kernel module (hardware virtualized). For directed ballooning, the guest must also have a few xenstore tools installed; in this case, memory statistics (proc/meminfo and /proc/vmstat) are communicated via xenbus to domain0 and balloon size can be calculated and controlled by domain0. We have not yet implemented a directed ballooning management tool in domain0 as selfballooning seems to be working very nicely so far.

Testing

To test xenballoond, we wrote a synthetic workload called "eatmem" that malloc's a random-sized chunk of memory (up to 512MB) and then writes to each page repeatedly a random number of times, then frees the memory and sleeps for a random number of seconds (up to 63). Then this sequence is repeated forever and we use /etc/rc.local to invoke this workload automatically for the lifetime of the domain. We also modified eatmem to use xenbus to communicate how large the chunk of memory is so that we can monitor it from domain0. Note that each guest is assumed to be configured with an adequately sized (virtual) swap disk.

On the domain0 side, we wrote a script to launch domains sequentially, watching memory usage to determine when it is OK to launch another one. Then we wrote another script to print certain xenbus keys for each domain and we use the "watch" command (with differences highlighted) to provide a xentop-like display to monitor these values. (See Figure 1.) Guests can be observed doing some swapping when eatmem suddenly requires the balloon to be rapidly deflated, but the phenomenom is transient and infrequent.

Though this is far from an industrial-strength test scaffold, it was sufficient to verify that selfballooning works, that metrics can be monitored from domain0, and that Linux was capable of handling random dynamic large inflation and deflation of the balloon. We ran this on two different 2GB machines running OracleVM (one an older laptop) and on each successfully launched seven pvm domains which were configured with memory=512 and maxmem=1024. On another test with more aggressive balloon inflation assumptions, we were able to launch fourteen 512MB idle domains.

Conclusions

While content-based page-sharing, VMM-based demand paging, and hotplug memory are all glamorous mechanisms that can be used to improve memory efficiency, the simple existing balloon driver provided by Xen, when combined with gray-box data collected by a few scripts, is sufficient to implement reasonable memory overcommit. More measurement and testing is ongoing in Oracle's OnDemand group, but we believe that this very simple solution delivers the vast majority of the value of memory overcommit with a much smaller cost.

References

1. J. Beulich, [PATCH] linux/balloon: don't allow ballooning down a domain below a reasonable limit,, Xen developers archive, http://lists.xensource.com/archives/html/xen-devel/2008-04/msg00143.html
2. P. Chen et al. When Virtual is Better than Real. In Proceedings HOTOS '01. http://www.eecs.umich.edu/~pmchen/papers/chen01.pdf
3. K. Fraser, RE: Memory Overcommit, Xen developers archive http://lists.xensource.com/archives/html/xen-devel/2005-12/msg00409.html
4. E. Horschman, Cheap Hypervisors: A Fine Idea -- If you can afford them, blog posting, http://blogs.vmware.com/virtualreality/2008/03/cheap-hyperviso.html
5. A. Kivity, Memory overcommit with kvm, http://avikivity.blogspot.com/2008/04/memory-overcommit-with-kvm.html
6. J. Kloster et al, On the feasibility of memory sharing: content based page sharing in the xen virtual machine monitor, Technical Report, 2006. http://www.cs.aau.dk/library/files/rapbibfiles1/1150283144.pdf
7. G. Milos, Memory COW in Xen, Presentation at Xen Summit, Nov 2007. http://www.xen.org/files/xensummit_fall2007/18_GregorMilos.pdf
8. J. Oneill, Expensive Hypervisors, a bad idea even if you can afford them. blog posting http://blogs.technet.com/jamesone/archive/2008/03/13/expensive-hypervisors-a-bad-idea-even-if-you-can-afford-them.aspx
9. J. Schopp et al, Resizing Memory with Balloons and Hotplug, Ottawa Linux Symposium 2006, https://ols2006.108.redhat.com/reprints/schopp-reprint.pdf
10. C. Waldspurger. Memory Resource Management in Vmware ESX Server, In Proceedings OSDI'02, http://www.usenix.org/events/osdi02/tech/walspurger/waldspurger.pdf
11. T. Wood, et al. Black-box and Gray-box Strategies for Virtual Machine Migration, In Proceedings NSDI '07. http://www.cs.umass.edu/~twood/pubs/NSDI07.pdf

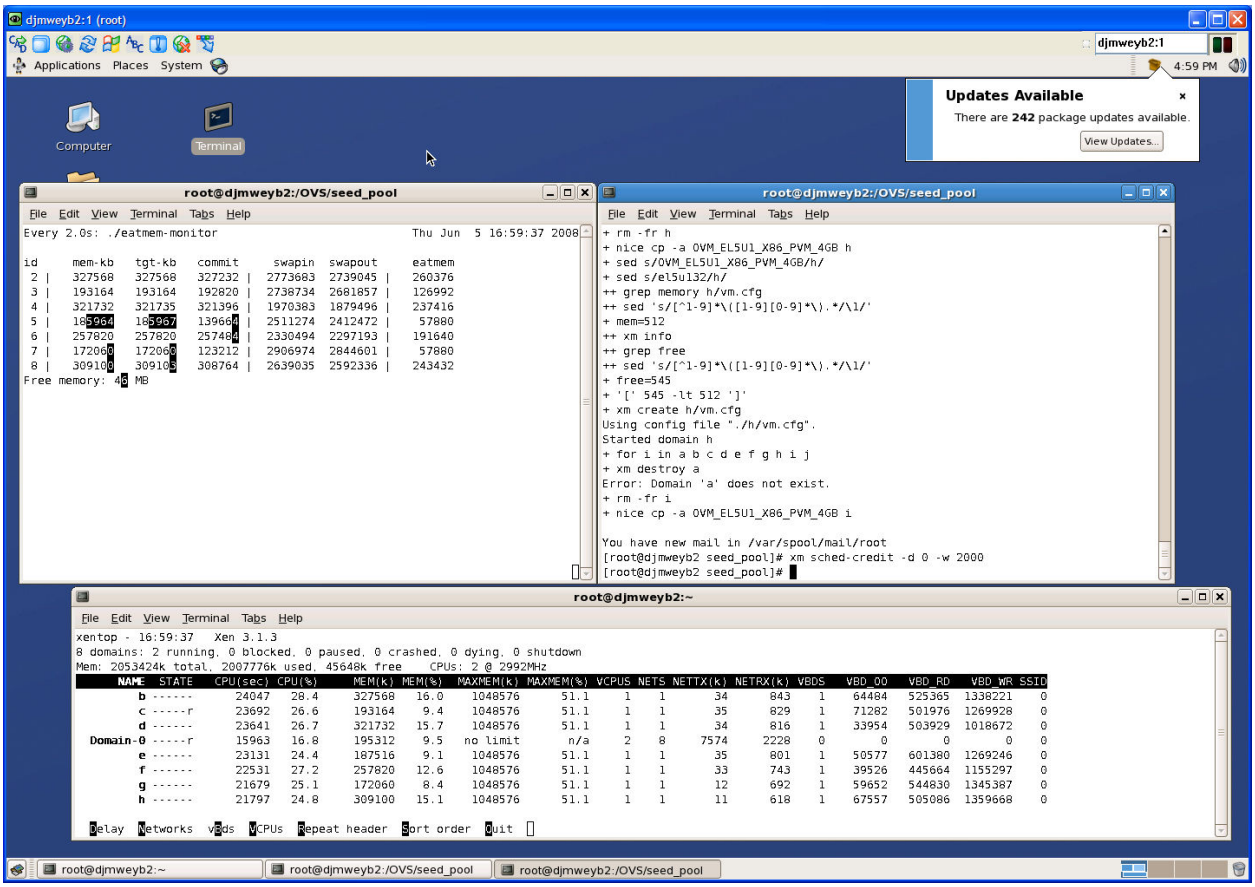

Figure 1. Memory-overcommited machine showing memory usage monitor for selfballooned guests